Journal of Computational Science 14 (2016) 51–60

Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

Exploiting data representation for fault tolerance

J. Elliott[a,b,*], M. Hoemmen[b], F. Mueller[a]

[a] North Carolina State University, Department of Computer Science, Raleigh, NC, United States
[b] Center for Computing Research, Sandia National Laboratories, Albuquerque, NM, United States

ARTICLE INFO

Article history:
Received 15 July 2015
Received in revised form 9 November 2015
Accepted 10 December 2015
Available online 6 January 2016

Keywords:
Algorithm-based fault tolerance
Resilient algorithms
Numerical methods

ABSTRACT

Incorrect computer hardware behavior may corrupt intermediate computations in numerical algorithms, possibly resulting in incorrect answers. Prior work models misbehaving hardware by randomly flipping bits in memory. We start by accepting this premise, and present an analytic model for the error introduced by a bit flip in an IEEE 754 floating-point number. We then relate this finding to the linear algebra concepts of normalization and matrix equilibration. In particular, we present a case study illustrating that normalizing both vector inputs of a dot product minimizes the probability of a single bit flip causing a large error in the dot product's result. Furthermore, the absolute error is either less than one or very large, which allows detection of large errors. Then, we apply this to the GMRES iterative solver. We count all possible errors that can be introduced through faults in arithmetic in the computationally intensive orthogonalization phase of GMRES, and show that when the matrix is equilibrated, the absolute error is bounded above by one.

1. Introduction

In the field of high-end computing (HEC) the notion of reliability has tended to focus on keeping thousands of physical nodes operating cooperatively for extended periods of time. As chip manufacturing and power requirements continue to advance, soft errors are becoming more apparent [1]. This implies that reliability research must address the case that the machine does not crash, but that outputs during computation may be silently incorrect. There have been many studies into hardening numerical kernels against soft errors, that is, the researchers attempt to preserve the illusion of a reliable machine by detecting and correcting all soft errors. We take a more analytical approach. Instead of focusing on detection/correction, we seek to study how the data operated on impacts the errors that we can observe given soft errors in data — called silent data corruption (SDC).

The driving motivation behind our work is the uncertainty surrounding the reliability of an exascale-class machine [2–4]. We attempt to avoid speculation over what hardware may be used in future (or present) HEC deployments, and instead analyze how a single soft error in an IEEE-754 floating-point number behaves. It has already been shown that existing and decommissioned HEC deployments have suffered from SDC [1,5]. For the prior reasons, we seek to study the link between the data operated on and soft errors. We intentionally perform our research subject to the IEEE 754 specification, which we believe will be used regardless of the architecture. We also restrict our analysis to single bit flips. This gives us a base line from which to draw higher-level conclusions related to multiple bit flips, and lets us isolate the impact of a bit flip.

IEEE 754 both defines the binary *representation* of data, and bounds the *rounding error* committed by arithmetic operations. This work focuses on data representation. The effects of rounding error on numerical algorithms, including those studied in this paper, have been extensively studied (see, e.g., [6]). However, these results generally only apply to *small* errors, such as those resulting from rounding. The error from bit flips can be huge and thus require different methods of analysis, like those presented in this paper.

2. Related work

Researchers have approached the problem of SDC in numerical algorithms in various ways. Many take the approach of treating an algorithm as a black box and observing the behavior of these codes when run with soft errors injected. Recently, Howle et al. [7] analyzed the behavior of various Krylov methods and observed the variance in iteration count based on the data structure that experiences the bit flip. Shantharam et al. [8] analyzed how bit flips in

* Corresponding author at: North Carolina State University, Department of Computer Science, Raleigh, NC, United States.
E-mail addresses: jjellio3@ncsu.edu (J. Elliott), mhoemme@sandia.gov (M. Hoemmen), mueller@cs.ncsu.edu (F. Mueller).

http://dx.doi.org/10.1016/j.jocs.2015.12.002
1877-7503/

a sparse matrix-vector multiply (SpMV) impact the L^2 norm and observed the error as CG is run. Bronevetsky et al. and Sloan et al. [9,10] analyzed several iterative methods documenting the impact of randomly injected bit flips into specific data structures in the algorithms and evaluated several detection/correction schemes in terms of overhead and accuracy. Exemplifying the concept of black-box analysis, BIFIT [11] characterizes applications based on their vulnerability to bit flips. A similar tool, S-FETI [12] injects soft errors through an emulation layer, allowing arbitrary codes to be run in a faulty virtual environment. Rather than focusing on how to preserve the illusion of a reliable machine or devising a scheme to inject soft errors, we investigate an avenue mostly ignored, which is how the data in the algorithm can be used to mitigate the impact of a bit flip.

Hoemmen and Heroux proposed a radically different approach. Rather than attempt to detect and correct soft errors, they use a "selective reliability" programming model to make the algorithm converge *through* soft errors [13]. Sao and Vuduc showed that reliably restarting iterative solvers enables convergence in the presence of soft errors [14]. In the same vein, Elliott et al. showed that bounding the error introduced in the orthogonalization phase of GMRES lets FT-GMRES converge with minimal impact on time to solution [15]. Boley et al. apply backward error analysis to linear systems in order to distinguish small errors due to rounding from inacceptably large errors due to transient hardware faults [16]. In general, our work complements this line of research. While Elliott, Hoemmen, and Sao have investigated algorithms that can converge through error, we show that in certain numerical kernels the data itself can have a "bounding" effect. For example, coupled with the work of Elliott et al. [15], we improve the likelihood that errors fall within the derived bound.

3. Motivation

Bit flips are the motivation for resilient algorithms. Algorithms run on digital machines, so any corruption is due to a fault at the hardware level or radiation from space. There are many layers and abstractions between what a user sees and what hardware actually does. Caches for data and instructions are an example of these layers.

Many works in the field of algorithmic fault tolerance have tailored methods for a bit flip fault model, e.g., [10,17–21], but their fault model rarely considers faulty hardware. Instead, researchers simulate hardware faults by injecting bit flips into values before or after an operation. A standard approach, e.g., [9,14,22], is to flip one or more bits in the input to an operation and observe the effect on algorithms.

We use this same fault injection methodology, but relate our findings to the analytic model of IEEE-754 floating point representation. Our findings demonstrate the close relationship between data and a fault model. Moreover, our findings illustrate the shortcomings of evaluating algorithms using this type of fault injection. As we will demonstrate, the expected value obtained by sampling is highly dependent on the characteristics of the data operated on. While bit flips are the root cause of errors, it is not clear that we need to assess algorithmic fault tolerance approaches using a bit flip fault methodology.

4. Overview

To explore the relation between data representation and soft errors, we first construct an analytic model of a soft error in an IEEE-754 floating-point scalar, and then extend this to a dot product. We uncover through analysis that the binary pattern of the exponent can be exploited for fault tolerance. We show this graphically via a case study using Monte Carlo sampling of random vectors, and then extend the idea of data scaling to matrices by using sparse matrix equilibration. To demonstrate the feasibly and utility of our work, we analyze the GMRES algorithm and instrument the computationally intensive orthogonalization phase. We count the possible absolute errors that can be introduced via a bit flip in a dot product, and show that scaling data lowers the likelihood of observing large, undetectable errors.

This paper is organized as follows:

1. In Section 5, we construct an analytic model of the absolute error for single bit upsets in IEEE-754 floating-point numbers.
2. In Section 6, we extend our model of faults in IEEE-754 scalars to vectors of arbitrary values, and present examples of how data scaling impacts the binary representation and absolute error we can observe.
3. In Section 7, we sample random dot products and compute the probability of having an error larger than one using Monte Carlo.
4. In Section 8, we link data scaling to sparse matrix equilibration, and instrument and evaluate the impact of a soft error in the computationally intensive orthogonalization phase of GMRES.

5. Fault model

The premise of our work is that a silent, transient bit flip impacts data. Before we can perform any analysis or experimental work, we must define how such a bit flip would impact an algorithm, and how we enforce that the bit flip was transient. To achieve this goal, we build our model around the basic concept that when an algorithm uses data, this translates into some set of operations being performed on the data. Should a bit flip perturb our data, some operation will use a corrupt value, rather than the correct value. The output of this single operation will then contain a tainted value, and this tainted value could cause the solution to be incorrect. Note that a transient bit flip may cause a persistent error in the output depending on how the value is used.

A side benefit of an operation-centric model is that we naturally avoid a pitfall to which arbitrary memory fault injection succumbs, namely that if a bit flip impacts data (or memory) that is never used (read) then this fault *cannot lead to a failure*. Our fault model allows a bit flip to perturb the input to an operation performed on the data, while not persistently tainting the storage of the inputs. This mimics how a transient bit flip would manifest itself, e.g., during ALU activities. As a result, the data that experiences the bit flip need not show signs that it was perturbed. This model allows us to observe the impact of transient flips on the inputs, which results in sticky or persistent error in the result. We then utilize mathematical analysis to model how this persistent error propagates through the algorithm.

5.1. Fault characterization

To derive a fault model we must first understand what a fault is. Since floating-point numbers approximate real numbers and most numerical algorithms use real values, we start from the definition of a real-valued scalar $\gamma \in \mathbb{R}$. The range of *possible* values that γ can take is $\gamma \in [-\infty, +\infty]$. We assume that the IEEE-754 specification for double-precision numbers, called *Binary64*, is used to represent these numbers. This means that γ can take a fixed set of numeric values. The range of $|\gamma|$, excluding 0 and denormalized numbers, is

$$|\gamma| \in [1.0 \times 2^{-1022}, 1.\bar{9} \times 2^{1023}]; \quad (1)$$

where $1.\bar{9}$ indicates the largest possible fractional component, and 1.0 indicates the smallest fractional component.

To approximate real numbers, *Binary64* uses 64 bits, of which 11 are devoted to the exponent, 52 for the fractional component